The STAR offline framework

V. Fine [a,b] Y. Fisyak [a] V. Perevoztchikov [a] T. Wenaus [a]

[a] *Brookhaven National Laboratory, Upton, New York 11973, USA*

[b] *Joint Institute for Nuclear Research, Dubna, 141980, Russia*

STAR

Abstract

The Solenoidal Tracker At RHIC (STAR) is a large acceptance collider detector which started data taking at Brookhaven National Laboratory in summer 2000.

STAR has developed a software framework for simulation, reconstruction and analysis in offline production, interactive physics analysis, and online monitoring. It is well matched both to STAR's present status of transitioning from a Fortran/C++ base to a fully object oriented (OO) base, and to the emerging OO base. This paper presents the results of two years of effort developing a modular C++ framework based on the ROOT package. The framework encompasses both wrapped Fortran components (legacy simulation and reconstruction code) used via IDL-defined data structures, and fully OO components (all physics analysis code) served by a recently developed object model for event data.

The framework supports chronologically chained components, which can themselves be composite sub-chains, with components ("makers") managing "data sets" they have created and are responsible for.

Makers and data sets inherit from the TDataSet class which supports their organization into hierarchical structures for management. TDataSet also centralizes almost all system tasks such as data set navigation, I/O, database access, and inter-component communication.

This paper will present an overview of this system, now deployed and well exercised in production environments with 10 TBytes real and 3 TBytes simulated data, and in an active physics analysis development program.

Key words: OO, Fortran, C++, ROOT, dataset, hierarchy, STAR, RHIC
PACS: 29.85.+c 07.05.Kf 07.05.Rm

⋆ Expanded version of a talk present for Chep2000 (Padova, February, 2000)

1 Introduction

The new generation of HENP experiments such as those at RHIC and LHC must contend with processing and mining heretofore unprecedented amounts of data with highly complex analysis software developed and used by large worldwide communities of physicists. Object oriented (OO) programming has been identified and adopted by these communities as an efficient and powerful approach to developing capable, robust, maintainable software in this environment.

Two years ago STAR started to develop a software framework [1] supporting simulation, reconstruction and analysis in offline production, interactive physics analysis and online monitoring environments to support STAR's transition from Fortran to C++ based software and to support the fully OO software base to which STAR is migrating [2].

For the purposes of this paper we follow [3] in describing a software "framework" as "a set of cooperating classes that make up a reusable design for a specific class of software". The framework dictates the architecture of the application and defines its overall structure: its partitioning into classes and objects and the key responsibilities thereof; how the classes and objects collaborate; and the thread of control. The framework predefines these design parameters so physicists can design their solutions using a proven programming model and can concentrate on the specifics of their applications.

We understood that a framework would not simply materialize by using object-oriented techniques. The framework requires a lot of attention if it is going to be successful.

This paper presents STAR's C++ ROOT-based [4] class library [5] and STAR production framework.

2 STAR C++ ROOT-based class library

Basing our framework on ROOT allows us to re-use what ROOT does well, including what it inherits from 30 years of CERN Program Library experience on the part of its developers. Examples include the built-in C++ data dictionary, object I/O services, 2D and 3D visualization, graphical object browsing, a C++ like scripting language, and comprehensive run time type information (RTTI). Adopting ROOT enabled STAR to concentrate on functionality specific to STAR [1] [6] and/or lacking in ROOT.

The STAR framework supports chronologically chained components, which can themselves be composite sub-chains, with components ("makers") managing "data sets" they have created and are responsible for. The OO model of STAR reconstruction chain is described in terms of the TDataSet C++ class. This class is used to describe the hierarchy of the data as well as the hierarchy of the program flow control ("chain") of the reconstruction code modules.

Makers and data sets inherit from the TDataSet class. This allows them to be organized into hierarchical structures for management. TDataSet centralizes almost all system tasks such as data set navigation, I/O, database access, and inter-component communication.

The properties of the TDataSet class can be defined as follows:

***TDataSet* object ::= "named" collection of *TDataSet* objects,**

where the "collection" (a pointer to a ROOT collection object) may contain no object.

An instance of the TDataSet class has constituent 'name' and 'title' (description) character strings, and may have a back pointer to its 'parent' TDataSet.

The class is used to implement directory-like data structures and maintain them via the TDataSetIter iterator class. TDataSet can be iterated either by using TDataSetIter or by the "TDataSet::Pass" method (see below). It allows us to introduce and change the following relationships between its components:

- *Data Set Member.* Any TDataSet collection member is called a "Data Set Member".
- *Structural Member.* A Data Set Member is a "Structural Member" if its back pointer points to the containing TDataSet object.
- *Data Set Owner (parent).* A TDataSet object "owns" (is an owner or parent of) any TDataSet object which is one of its "Structural Members".
- *Associated Member.* A collection member which is not a "Structural Member" of the containing data set is said to be an "Associated Member" of the data set.
- *Orphan Data Set.* If a data set is not a member of any TDataSet object it is termed an "orphan" data set object.

Fig. 1 shows the class diagram of STAR classes derived from TDataSet to build the OO model of STAR's offline framework.

TDataSet provides services which help the user to build and manage a hierarchy of his or her data. For example, the method TDataSet::Update allows one TDataSet (main) object to be updated with another TDataSet (donor). The"TDataSet::Update" method matches the structure of the "main" and

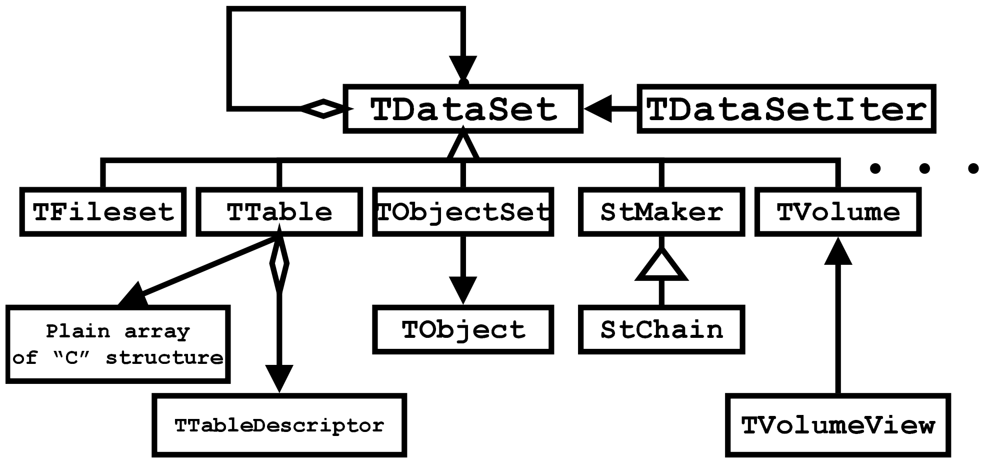

Fig. 1. UML class diagram of STAR simulation / reconstruction chain

"donor" data sets, and replaces the matched branches of the main data set with ones from the "donor". If the "donor" set contains branches which are not present within the main one, they are added to the main data set. This is useful when only a fraction of the entire structure has to updated within a chain looping over "events". The method TDataSet::Shunt allows changing the relationship of a particular TDataSet object with others.

2.1 Classes StChain/StMaker

StMaker is a base class to describe a single processing step (termed a "maker") of the chain of makers controlled by an object of the StChain class, where StChain is both a subclass of StMaker and the principal driver for the processing chain. Makers comprise the modular processing components that are chained together into processing jobs. The instances of StMaker comprise a dedicated branch of data sets. They have access to the rest of the data set tree and can use the legacy Fortran/C simulation and reconstruction modules via special "module wrapper" classes [7]. "StMaker" is derived from TDataSet.

The functions of StMaker are to

- perform a maker's job initialization with the StMaker::Init method;
- perform event level processing, and create and own the TDataSet structure containing processing results, via the StChain::Make method;
- provide its own data for other makers by adding it to the public chain with the StMaker::AddData method;
- prepare the internal data structures to accomodate the next event with the StChain::Clear method;
- perform a maker's job close-out with the StMaker::Finish method.

Typically, the only mandatory method the user code must overload is StMaker::Make.

The StChain class supports the definition and control of an analysis job defined as a chain of makers which perform the individual processing steps in the job.

By design, each concrete maker class is supplied from a dedicated shared library which is to be loaded at run time. The steering code appropriate to a desired chain of components defining the needed analysis job can be specified, loaded and customized rapidly and efficiently at run time.

Each instance of StMaker included in an StChain represents a modular component implementing a processing step and typically generating and making available data and control/bookkeeping information. In our framework, any part of the chain (thanks to TDataSet inheritance) can be easily isolated by the steering code and its data saved and restored using the various supported I/O formats. This is possible because each TDataSet branch of the entire TDataSet structure can be written out and read back independently.

2.2 Classes TTable, TTableSorter, TTableIter

The TTable class is a base class wrapper to maintain arrays of plain C structures. These arrays are called (and effectively are) tables, with columns and rows.

The initial purpose of this class was to provide STAR with a tool to migrate from StAF (Standard Analysis Framework), a framework employed in an early stage of the project [2].

A table is a very convenient form in which to save and manipulate huge amounts of experimental data. All known databases can manipulate tables. It is much more simple to resolve "schema evolution" problems with "plain" tables ("plain" means that the arrays contain no pointers to other objects/structures). Tables can be used easily in a mixed code environment (Fortran/C/C++).

A TTable instance is generated for each table and is supported by two companion classes: TTableSorter to sort table rows by various foreign keys, and TTableIter to loop over the sorted table rows.

The present STAR framework provides several I/O formats to save and restore these objects. There are XDR-based (XDF), ROOT, plain ASCII and MySQL formats. The I/O format is chosen by steering code. The user's code is in general unaware of the I/O format and is not affected when the format changes or a new I/O schema is introduced.

Since TTable is a subclass of TDataSet it is easy to combine instances in the various hierarchical structures (Fig.2). In this way we compensate for the lack of pointers within TTable objects. Fig. 2 shows how one can create a rather complicated object and access it interactively via point-and-click interface provided by the ROOT object browser.

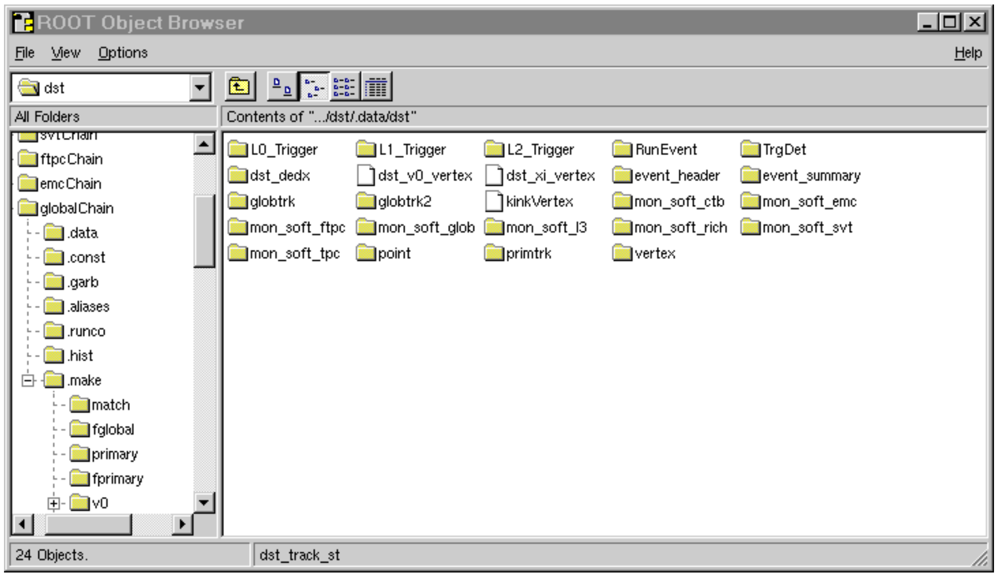

Fig. 2. STAR data structures viewed via ROOT object browser

2.3 Class TFileSet

Class TFileSet is to map a real "file directory tree" of the platform we are running on to the TDataSet instance in memory. Each "pure" TDataSet object is related to a real directory of some file system (UNIX or Windows for example). The extra data members of subclasses of TDataSet (TTables for example) can be converted into an ASCII representation, a ROOT macro, with the ROOT-provided TObject::SavePrimitive method. This approach allows people to get access to any piece of what is really a "naive" database; to create, edit and play with data in the same way people do with software code. Any user can easily prototype a database they may have in mind. A user of the STAR framework "by definition" has basic skills in using C++ classes like TDataSet and TTable and in common shell commands. Designing such a "naive" database doesn't require any extra knowledge or special skill.

Everyone is free to use for their particular task either the "central" data base – the "CVS repository" – or one can "check out" material from CVS and adjust it for current needs with trivial shell commands like "mkdir" / "rmdir" and a plain text editor.

This class together with standard CVS and AFS facilities allows creation of a “naive” but powerful database. AFS provides world-wide access and CVS supplies the “house-keeping”.

2.4 Classes TVolume/ TVolumeView

The TVolume and TVolumeView classes provide tools to access the detailed detector knowledge implemented in a GEANT3 simulation model [8]. Instances of this class are in use by various pieces of the STAR simulation/reconstruction chain, and provide the detector geometry for “Event Display” classes (see: Fig. 3).

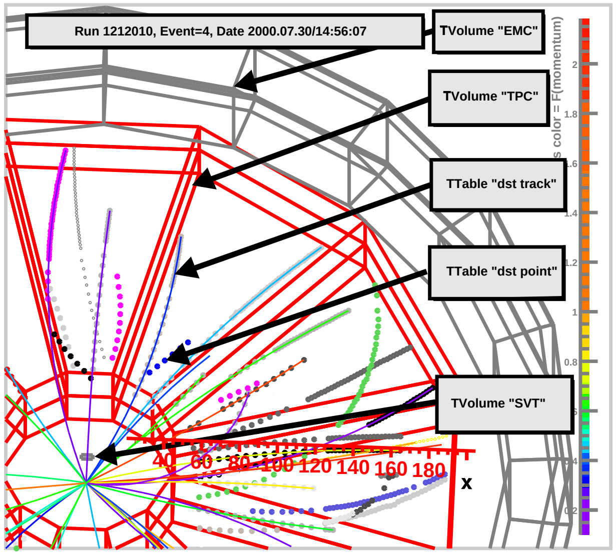

Fig. 3. A snapshot of the ROOT-provided “view” of STAR data set objects combining TVolume (detector geometry) and TTable (event data components) objects

3 Conclusion

Today the STAR reconstruction chain is defined as a single instance of the StChain class. It holds instances of up to 84 subclasses of StMaker. All together they create about 300 different instances of TTable involving 156 legacy Fortran/C modules.

The table shows how the usage of the C++ code vs Fortran has changed over one year.

The present framework has allowed the construction of hierarchical organizations of components and data, and centralizes almost all system tasks such as

Table 1
Trend in usage of OO technology in the STAR offline software since CHEP'98 (in kLoc (kLoc = 1000*(Line-Of-Code)))

Language	**CHEP 98**	**CHEP 2000**	**2000/98**
C++	27	138	5.1
FORTRAN	90	68	0.75
MORTRAN	28	34	1.2
C	20	24	1.2
IDL	10	13	1.3
ROOT macros	-	11	-
KUIP	22	4	0.2
Total kLocs	**197**	**292**	**1.5**

data set navigation, I/O, database access, and inter-component communication. The framework has supported the replacement of legacy Fortran codes with new C++ OO codes. Certain special languages are still in localized use for specialized applications; for example, MORTRAN is in use to define the complex detector geometry in the absence of any satisfactory replacement [8].

The framework was adopted in 1998 as the official STAR offline framework. Following extensive testing and development it has been used to produce 3.3 TBytes of reconstructed data from the 10 TBytes of real raw data taken during the first physics run.

All classes presented in this paper have been included in the standard ROOT package. Sources and HTML documentation are available for download from the official ROOT Web site: `http://root.cern.ch`.

4 Acknowledgements

This research was supported by the U.S. Department of Energy under Contract No. DE-AC02-76CH00016.

References

[1] R.Bossingham, et al. STAR Offline Simulation and Analysis Software Design, STAR Note. No. 281, January, 1997 (Brookhaven National Laboratory, 1997)

[2] V.Fine, et al. Steps Towards C++/OO Offline Software in STAR, in: CHEP'98, (`http://www.hep.net/chep98/181.html`) Chicago, Autumn 1998.

[3] Erich Gamma, et al. Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Pub Co, 1995)

[4] N.Buncic, R.Brun, V.Fine, F.Rademaker, ROOT: An Object-Oriented Framework, Proceeding of "AIHENP'96 Workshop", Lausanne, 1996.

[5] V.Fine, STAR C++ ROOT-based class library, in: US HENP ROOT Workshop, Chicago, USA, March, 1999.

[6] Y.Fisyak, ROOT in STAR. in: US HENP ROOT Workshop, Chicago, USA, March, 1999.

[7] V.Perevoztchikov, G.V.Buren STAR Maker Schema, in: (`http://www.star.bnl.gov/STAR/html/comp_l/root/MakerSchema/index.htm`) BNL, New York, USA, 1999.

[8] V.Fine, P.Nevski, OO model of STAR detector for simulation, visualization and reconstruction, in: CHEP'2000, Padova, Italy, 2000.